What we learned from assignment 2

We realised towards the end of our alpha, that the disabilities we wanted to support were mostly already covered by existing Android system settings. Everything from high contrast modes to larger text options. This meant we needed to think about which disabilities weren't already supported, but also how we could design our app to work well with existing features. This was a key theme for our beta.

Planning & Team Communication

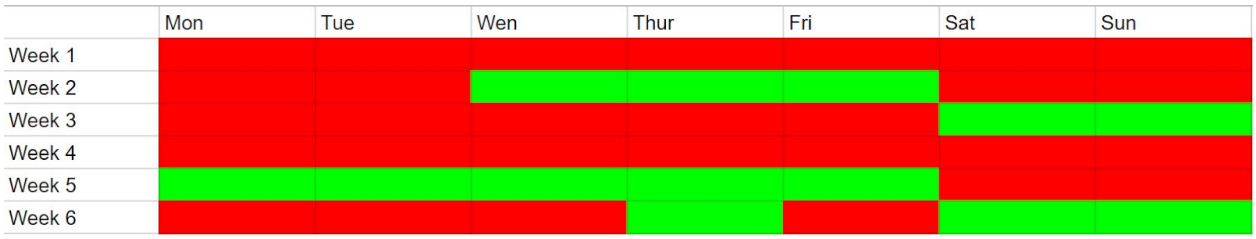
We had 6 weeks until our beta was due, and so we split each feature we wanted in our beta into programming tasks and assigned a difficulty to them, aiming for around 15 points each week.

Java
HistoryFragment + loader (10)
Navigation/UI (10)
SettingsFragment(10)
NotificationHelper (10)
AlarmReceiver(10)
Shared preferences(10)

XML
HistoryFragment.xml(3)
SettingsFragment.xml(3)

Other
Colour deficiency options(2)
Comments(3)
Bugs(3)

Just like before, we used a shared Google Sheets document to indicate the days we were free, if a single person wasn't free on a particular day, we would highlight those days red. We aimed for at least three days a week to meet up and work on the assignment together, roughly planning which parts of the app we wanted to work for that day.



	Mon	Tue	Wen	Thur	Fri	Sat	Sun
Week 1							
Week 2			History + Nav (20)	History + Nav(20)	History xml (3)		
Week 3						NotificationHelper (10)	AlarmReceiver(10)
Week 4							
Week 5	AlarmReceiver(10)	AlarmReceiver(10)	AlarmReceiver(10)	SettingsFragment + xml (13)	Shared preferences (10)		
Week 6				Colour settings (3)		Debugging/Comments	Comments

In our alpha we worked together during all stages to avoid conflicts and knowledge discrepancies, we employed a mostly similar strategy for the beta, but with a key difference in how we divided research and testing responsibilities. This was due to our app becoming more complex and our time this semester being more limited. Jerry was assigned with designing color blind friendly colour schemes, while Jaydin worked on the pseudo code for the history fragment and Ryan worked on receivers and system settings/preferences. We each worked in our own sandbox (see testing section for more info) that corresponded to the feature we were assigned. We would then meet up each week to share our knowledge and collaborate on implementing the features we had been testing in the sandbox, into the Flexi Task app. This way we could spend our limited time together working on our app, rather than reading android documentation as we already had experience working with that feature.

Disability Support

Here are the different options we are aiming to support, with the goal of enabling users with these disabilities to utilise our app just as effectively as users without these disabilities:

- Vision impairments:
    - Content descriptors: We continue to use content descriptors in most parts of our app (aside from the flexi and fixed timelines). These are tested manually by turning on the screen reader accessibility options in the emulator. These will be tested more thoroughly for the final release.
    - Large touch targets: We've carefully followed Android's recommended size conventions to easily allow users with different magnification settings to use our app as effectively as users without this setting. We've tested this on various screen devices in emulation with a group member who uses these settings and has minor vision impairment.

- Colour deficiency: Android phones come equipped with settings that let users with color deficiencies change how colours appear on their screens. While this option is great, it is just a system level setting, therefore it doesn't guarantee that the colour schemes for each individual app will look good or keep in line with standard UI colour conventions. For our beta we decided to create custom colour deficiency themes/colour modes that would be viewable to a person with a specific colour impairment while also acting as a

viable alternative theme for someone without any kind of colour deficiency issues. Currently, we support the most common form of colour deficiency, deuteranomaly, which is a reduced sensitivity to green light. We also support the slightly less common forms called Protanopia and Tritanopia, and are looking into more options for a final release.

- Memory impairment: Task reminder apps are useful for forgetful individuals, but are even more beneficial to users suffering from memory impairments. Our app now includes a history page that lets the user keep track of previously completed tasks, as well as a daily notification to remind the user of upcoming tasks. We are currently working on more settings for this notification, including a possibility of individual notifications per task (although this is a battery management issue we will have to weigh up).

General Testing & Accessibility Testing

We created a strong foundation for our app in the alpha stages, we had a working SQL database plus all UI components were functional, and so the beta was all about implementing additional features rather than massive functionality overhauls. Normally to do this you would branch off from the master-git so these new features wouldn't ruin your current working implementation. But because we were beginner Android developers, we found creating simple App versions of the feature we wished to implement helped us understand the various classes and how they would interact with our main app better - rather than getting lost in the complexities of our own app.
When it came time to implement Notifications we created a very simple app that would set up a alarm/notification when you clicked a button. For our new history fragment we tested a few different layouts in a seperate app, until we settled for a layout that wouldn't obscure any of the UI and would respond quickly to touch input. And for settings page we tried extending a dialog preferences to use our existing timepicker class (as our app's support libraries didn't support this), but after much trial and error we instead settled for a simpler StackOverflow solution.

We also tested how our UI would respond to accessibility changes, including text size changes, by using a group member who suffers from minor visual impairment. They said they were satisfied with how the app responded to their phone's accessibility settings.

To make sure we were using colour schemes that would be appropriate for colour blind individuals we used Colour Hexa to discover colours that were generally opposite each other on the colour wheel, but would also be visible to colorblind users. Once we had these colours down, we then implemented these in the settings of our app and used PNGs of our UI in the colour blindness simulator to simulate how they would appear to a colour blinded individual (see images at the end of the document). This is a far from a perfect solution, so we managed to contact two people, a close friend with Deuteranomaly and another with very mild Protanopia.The participant with Deuteranomaly liked the primary and secondary colours we used, but reported that the priority colours were not distinct enough, the other also complained

that they were hard to see. We will be working with them to find suitable colour scheme for the final release, the framework is all there, so it's just a matter of optimising the colour values in the colour resource file.

To test our disability support for people with memory impairments, we needed to make sure the app would remind them of upcoming and overdue tasks for the week. To do this we manually simulated time progressions in the system settings to move the time closer to the notification triggering event. The format for the notification will change for the final release, including showing the user how long until a task is due and its priority, rather than just showing the title.

Current known issues in beta:

- Due date for FixedTask does not include the time, so tasks occuring on the same day won't be ordered (ie: task 11/02/18 @11am vs 11/02/18 @12pm)
- Custom "recurring days" selector layout not formatted correctly for smaller devices
- Title of task overlaps with date due when long enough
- At various points in our app, we don't use the contentResolver to update the task and instead access our database directly via a raw query. This just makes it easier to debug as we are able to log and check each individual query. In our alpha we were originally having issues interfacing with our database via our content provider, but fixed it.
- When the user rotates the screen, the UI resets itself. This is a lifecycle problem with our app, which we will address before final release.
- Status bar colour doesn't change, it remains blue, and Navigation bar colours don't change yet.
- When a user deletes a fixed task that hasn't been completed, the app incorrectly shows it in the history page.

Problems we found and fixed

The aforementioned small sandboxes helped tremendously when it came to debugging, for instance we discovered our notifications weren't properly firing on newer devices, this was due to a new API update needing you to create a separate notification channel for them to run on. We fixed this by checking which to see if what API the user's device used to determine whether or not we needed to create a channel. The Notification sandbox also let us play around with various alarm manager methods to see would work best for our app, our notification needed to be sent when the app was closed, so we found alarmmanager worked best as it sets an alarm at a system level and calls our Alert receiver even with our app shut down.

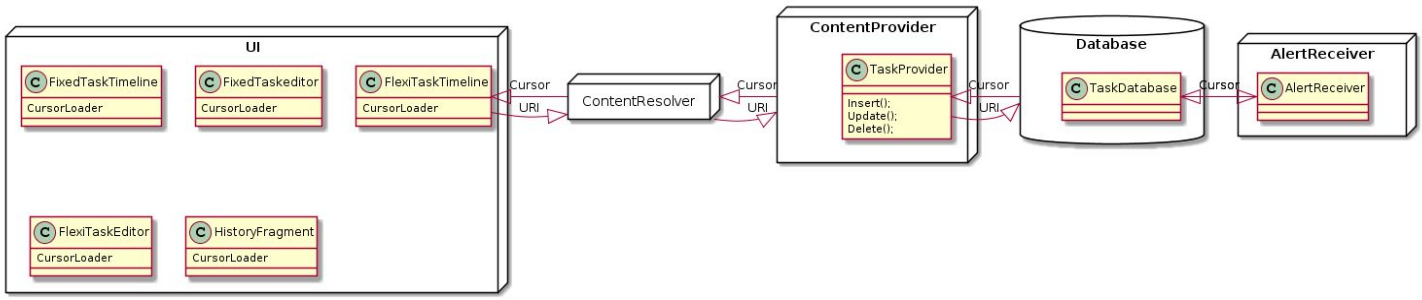
We needed to take everything we had and encase it in a new container that would allow the user to use a navigation drawer to navigate through all our different fragments. While this sounds simple enough, it wasn't, and broke a lot of the functionality we had. After we had solved the crashes, we had a problem with the appbar as it would cover up parts of the UI and didn't

work with the navigation drawer toggles and icons. This led us to dropping the default appbar and using a toolbar instead.

Version control

We started learning Github towards the end of our Alpha and as a result ran into complications and errors when it came time for submission. On the night before our assignment 1 was due, we managed to narrow down the errors to being an issue with authentication (the github credentials tied to Android studio differed from out Git Bash authentication). To fix this we deleted the Git folder for our app and started a fresh version control folder that we would use for our beta. To make sure we wouldn't repeat similar mistakes, we spent a couple of days over the mid semester break working through Udacity's Github tutorial, increasing our familiarity with Git conventions and Android Studio's implementation of it. Throughout our beta we would make sure to commit and push to the master.

How the app works



Our mainactivity (mainActivity) acts as a container to hold and switch fragments, based on what item the user has selected in its navigation draw. Our Timeline Fragment Container is selected by default and contains a viewpager containing the fragments for Fixed Timeline and Flexi Timeline, this viewpager allows the user to swipe between each timeline fragment. When these fragment classes are first created (onCreate() method is called) each class initializes its own loadermanager. Once this loadermanager is up and running, it calls the OnCreateLoader() method, which initializes a new CursorLoader. This cursor loader is given a projection (a string array of the columns the activity/fragment want), the URI (like a URL but for database tables) and a sortOrder (Flexitask wants the data returned to be sorted by its priority calculation, whereas fixed task it wants the data ordered by due date). This cursorloader in turn queries the contentresolver.

The contentresolver "resolves" the URI to our custom content Provider (Task Provider). This provider acts as an interface to the database, it's here that determines whether we are inserting,

updating, deleting or just querying data - and whether we want a specific row or the entire table. It performs the desired task and returns a Cursor with the data requested, back to the contentResolver, which returns back to the LoaderManager which passes it on to the cursorOnFinish() method. In this method we call the cursorAdapter swapCursor() method and pass it this newly returned cursor to process.

By using the loader thread and abstracting this data retrieval process to loaders and content providers, we allow the user to continue using the UI with few interruptions. We also use a similar querying technique in our editor activities, in addition to a simple if() statement that allows us to reuse the editor activities for both creation of a new task and updating an existing task.

We had to account for when the user “updates” the task from either the flexi or fixed timeline fragment in the form of the “done” button. When selecting “done” (the tick), for a fixed task, the app will check if that task is recurring or not. If it is it will update the tasks due date to (the current due date + recurring period * 86400000 (milliseconds in a day)) to get the new due date. It then restarts the cursor loader to go and retrieve these new changes.

When a flexi task is first created the date-last-completed field for that row/task is set to the current date (in milliseconds) and the next due date(in milliseconds) is derived from this by adding the recurring period * milliseconds in a day. When the user indicates they have finished an activity (by pressing the done button/ tick), the date-last-completed field is set to the current time and the next due date is recalculated.Our priority rating is calculated at two stages, one in when we query the data in the Flexitask fragment loader and want the return data sorted by priority and the other in the cursoradapter where we assign a XML color element to indicate the priority to the user.

This is the algorithm for the priority:

$$\frac{(\text{Todays Date} - \text{Date Created})/86{,}400{,}000 + 1}{\text{Recurring Frequency}}$$

This produces a number which determine how overdue a task is. We then organise the flexi tasks based on which tasks have the larger number, as well as assign priority colors (based on the user’s colour deficiency settings).

The HistoryFragment works much the same as the flexi and fixed task timelines, in that it uses a loader manager and cursor adaptor to obtain “deleted”(deactivated) tasks. The history fragment has several buttons and corresponding listeners, that act as filters, when a button is selected it restarts loader which retrieves the appropriate data using the new parameters.

When our app is started, our mainActivity creates a system alarm for a specified time (at first this is 8:00am, but can be changed in the settings). Once this alarm is triggered (eg: @8am), the Android system calls our AlarmReceiver class which cursors through upcoming active tasks, and adds their titles to an array. Once it has obtained these tasks, it utilises our Notification Helper class to create a Notification. This class first creates notification channels for Android devices running Oreo or higher to use, and creates a message with the given array of task titles from Alert Receiver. The result is returned to alert receiver and broadcasted to the users device. Alert receiver then sets up an alarm for the following day. We are currently working on a smarter implementation (App Manager) which sets the channel up the first time the app is started rather than every time the broadcast receiver calls the Notification Helper class.

Where to from here

Like before, we have met every single beta milestone we had set out. However due to the every increasing complex nature of our app, there are a lot more bugs this time round. We managed to fixed the main bugs that would crash our app or cause threading issues, but our UI doesn't work as well on devices with smaller screens and we're not processing our tasks correctly so the history fragment has some quirks. We also need to finalize our colour deficiency friendly themes, as well as look into Android Unit Testing to save us time manually changing the phones settings to simulate time progressions.

We're lucky to have reached a stage where we have the majority of our features list implemented, and so between now and our final release, we will be focusing on addressing the bugs and polishing the UI. With enough time, which we're sure we will have, we would like to look into an Android wearable version of our app and possibly look into implementing task labels and individual notifications for each task - these changes should be relatively easy to make (aside from the wearable app) as we already have most of the foundation there.

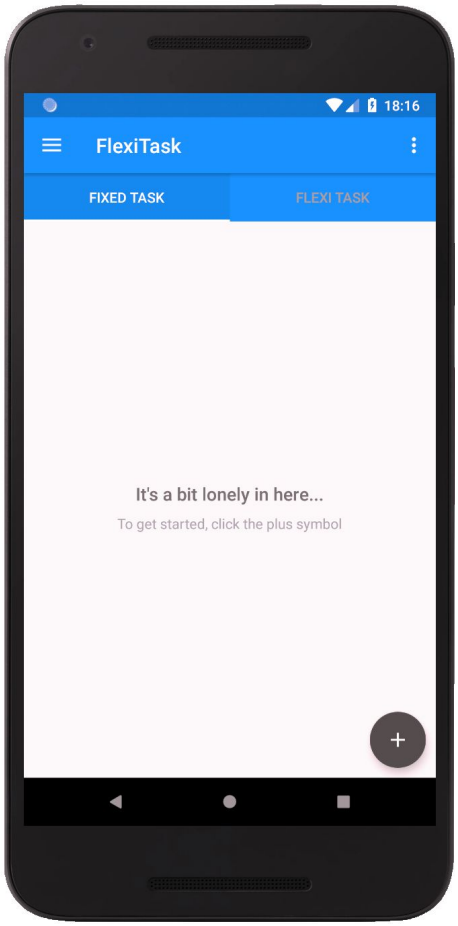
Overall, we remain happy with the progress we've made and feel optimistic about delivering what we outlined in assignment 1.

How to build our app

1) Download and install the latest version of Android Studio with the default settings (https://developer.android.com/studio/)
2) Download our project from Github
3) Unzip the file
4) Import our project into Android Studio (File->Import) or select the "open an existing Android Studio project" from the startup menu
5) Then browse to the FlexTaskBeta-master folder, make sure this folder shows an Android symbol
6) Once it's loaded (might take a while, as Gradle may need to download files), press the "RUN APP" button and select **Pixel** as the virtual device (the device we used for testing, but any device using an API higher than 19 should work) and click ok

NOTE:

- ***To test the different colour options, you must close and reopen the app, as the toolbar's colour can't be changed programmatically once inflated and requires a restart.***
- ***Notifications can be tested by setting a time in the settings (otherwise it should default to 8:00am) to be one minute ahead of the phone's settings.***
- ***The history section contains bugs at the moment due to how we're processing the task data and only displays tasks that the user explicitly deletes.***



When the corresponding theme is selected, this is what colour blind users will see (Deuteranomaly left, Protanopia right and Tritanopia bottom)